

CSE 634: Data Mining

Professor: Anita Wasilewska

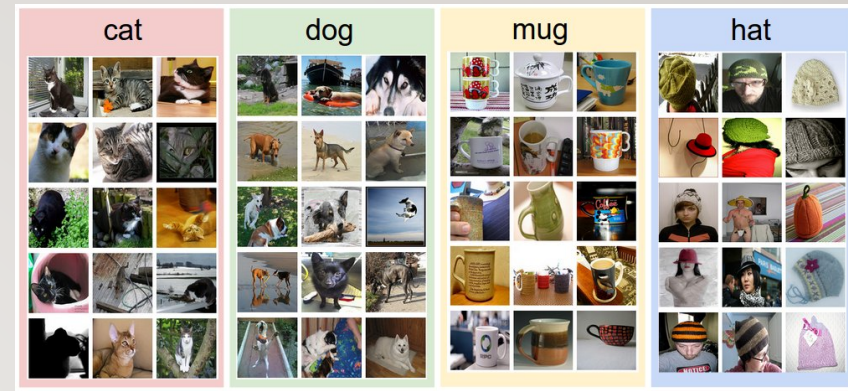


IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

REFERENCES

- <http://www3.cs.stonybrook.edu/~cse634/L7ch6NN.pdf>
- http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture2.pdf
- <https://deeplearning.web.unc.edu/files/2016/12/An-overview-of-gradient-descent-optimization-algorithm.pdf>
- <https://www.slideshare.net/infobuzz/back-propagation>
- <http://people.uncw.edu/tagliarini/Courses/415/Lectures/An%20Introduction%20To%20The%20Backpropagation%20Algorithm.ppt>
- <https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59>
- <https://cs231n.github.io>
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
- <https://medium.com/dbrs-innovation-labs/visualizing-neural-networks-in-virtual-space-7e3f62f7177>
- <https://www.kdnuggets.com/2016/06/visual-explanation-backpropagation-algorithm-neural-networks.html>
- <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/http://www.emergentmind.com/neural-network>
- https://en.wikipedia.org/wiki/Convolutional_neural_network

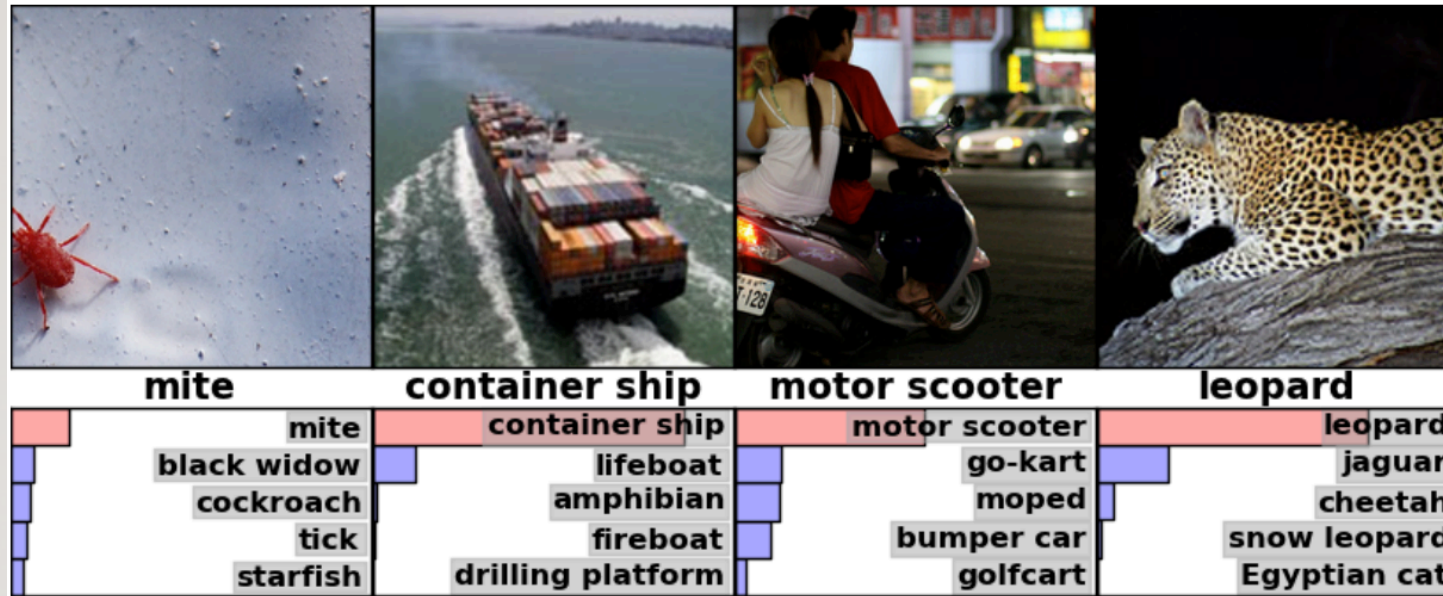
Paper

- Name: "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012
- Authors : Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton
- Conference: ILSVRC(ImageNet Large Scale Visual Recognition Competition)-2012

OVERVIEW

- Introduction to Image Classification
- Loss functions, Optimization and Gradient descent
- Neural Networks and Backpropagation Algorithm
- Convolutional Neural Networks
- Paper : Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

IMAGE CLASSIFICATION



Input: An image(matrix of pixel dimensions)

Categories/Labels : A set of pre-determined values which define an image.

Output: A label corresponding to the input image.

CHALLENGES

- Illumination:



- Deformation:



CHALLENGES

- Occlusion:



- Background Clutter:



INITIAL ATTEMPTS



Detect edges



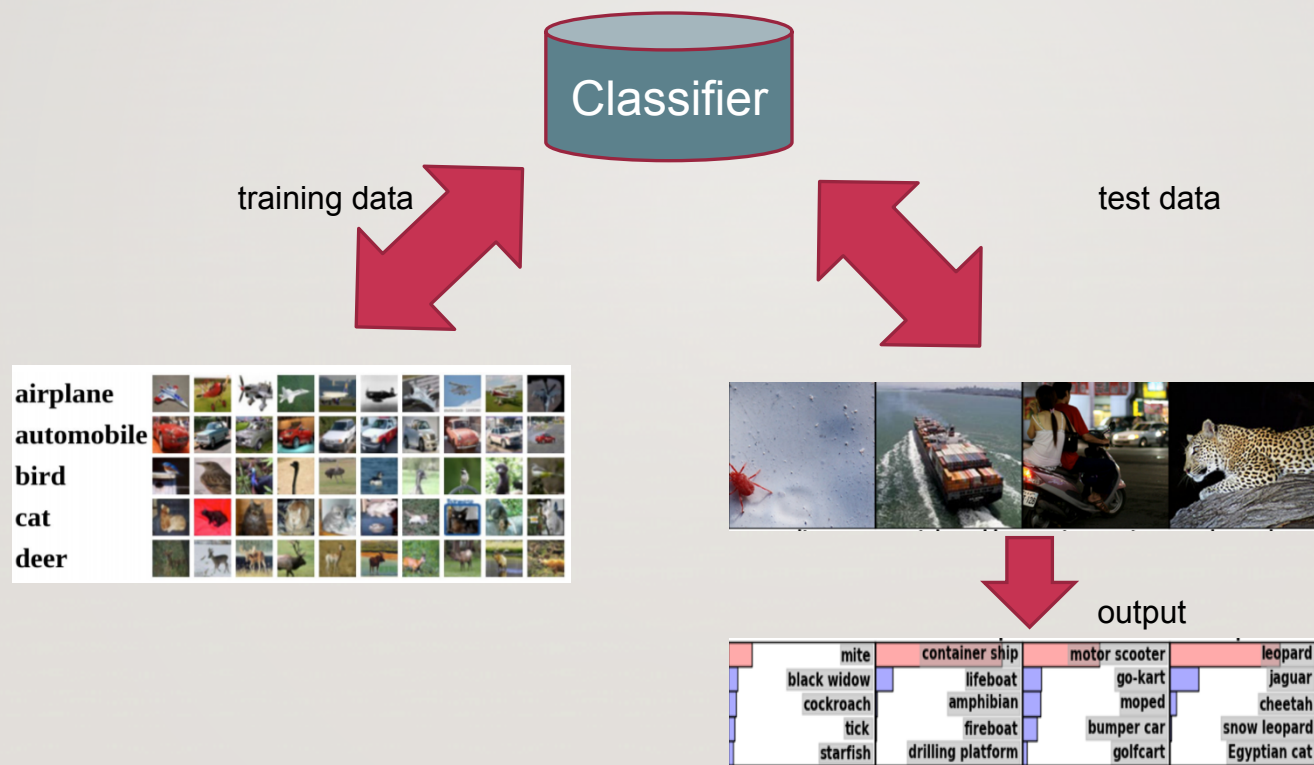
- Compute explicit “**Rules**” based on corners and boundaries and identify Labels based on these rules.

ex: Two lines meeting at a corner are a cat’s ears.

Pitfalls

- Time consuming, since we have start all over for an other object label.

A DATA DRIVEN APPROACH



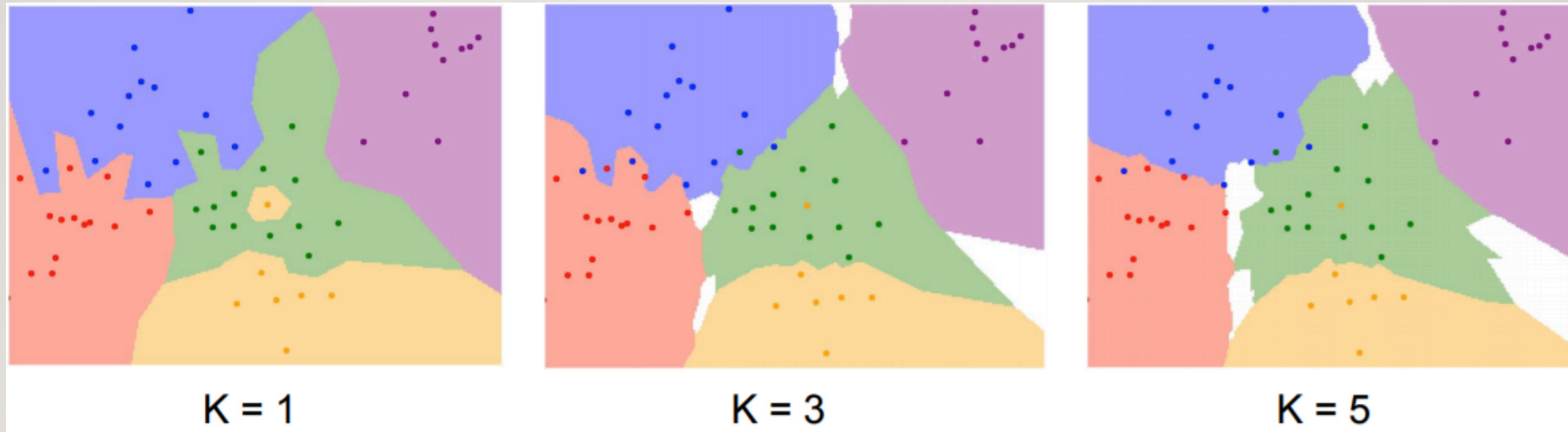
K-NEAREST NEIGHBORS

- Use a distance metric ex:L1 or L2 distance and compute the K-nearest neighbors i.e. K “**trained**” images having least difference of the distance metric from the chosen image.
- A majority vote is taken among the K neighbors and that is selected as the label of the test image.

L1 distance:
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image					training image					pixel-wise absolute value differences				
56	32	10	18		10	20	24	17		46	12	14	1	
90	23	128	133		8	10	89	100		82	13	39	33	
24	26	178	200	-	12	16	178	170	=	12	10	0	30	
2	0	255	220		4	32	233	112		2	32	22	108	add → 456

K-NEAREST NEIGHBORS



- Simply Memorize all training data and labels
- Choose a K on the training data and evaluate it on the testing data

Pitfalls

- Distance metric not very effective.
- Curse of dimensionality.

LINEAR CLASSIFICATION

A linear classifier is of the form

$$f(x,W) = Wx + b$$

x – Input vector $\{x_1, x_2, \dots, x_n\}$ where x_i is the value of a pixel dimension

W – set of weights assigned to each pixel dimension determined by the training data for each label.

b – bias for each label.

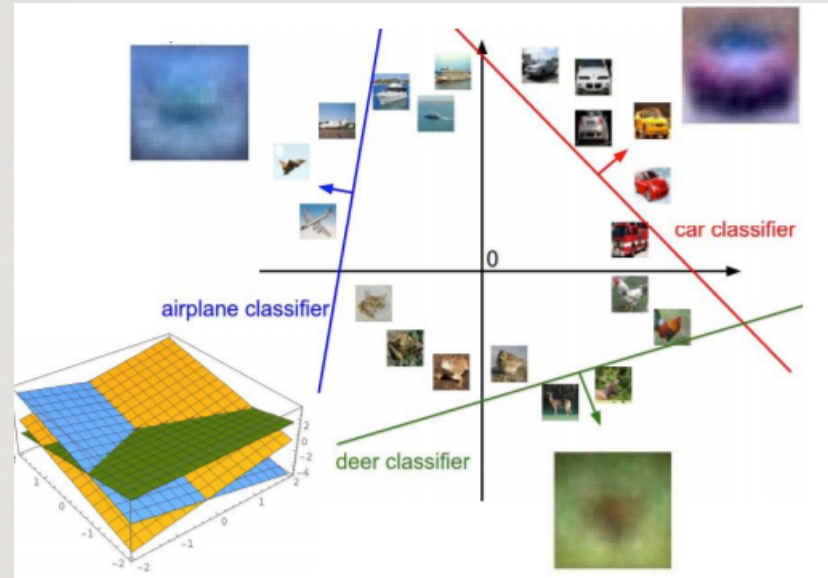
$f(x,W)$ – vector of scores corresponding to each label



$f(x,W)$

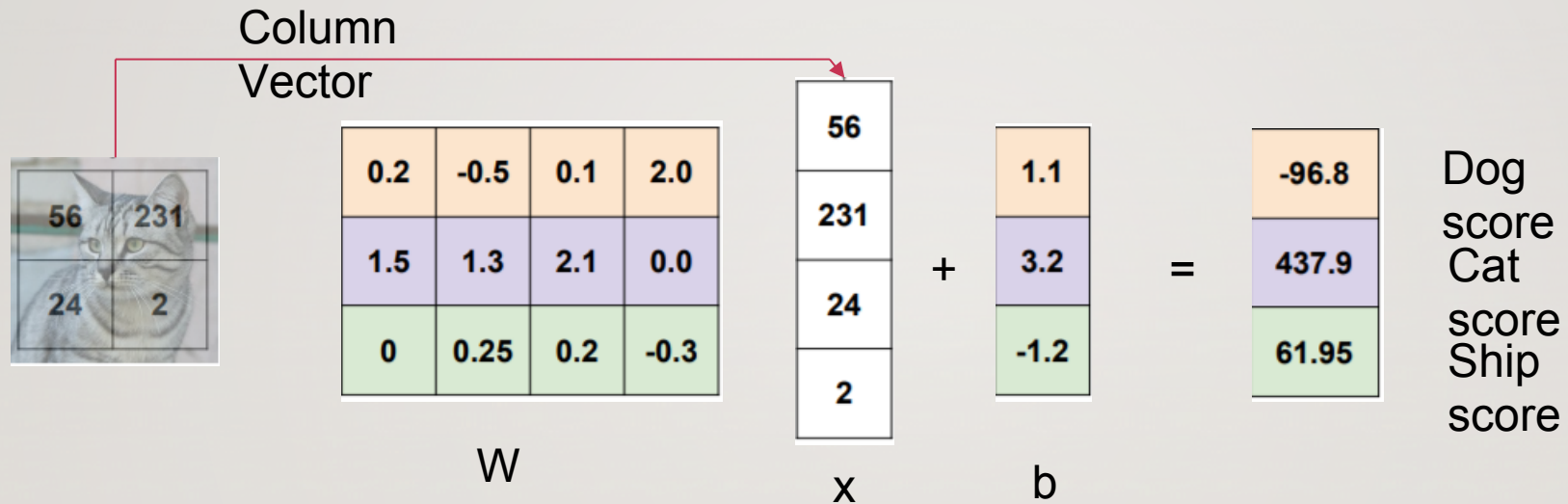
0.51	cat score
-1.09	dog score

INTERPRETING A LINEAR CLASSIFIER



- Each image is a point in the high dimensional space
- The linear classifier puts in the linear decision boundaries separating one category from the rest of the categories.

AN EXAMPLE



LOSS FUNCTIONS

- **Loss functions for classification** are computationally feasible loss functions representing the price paid for inaccuracy of predictions in classification problems (problems of identifying which category a particular observation belongs to).
- It describes how far off the result your network produced is from the expected result - it indicates the magnitude of error your model made on its prediction.

SUPPOSE: 3 TRAINING EXAMPLES, 3 CLASSES.

WITH SOME OF THE SCORES $f(x, W) = W_x$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1 \text{ to } N}$$

Where x_i is image and y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

SUPPOSE: 3 TRAINING EXAMPLES, 3 CLASSES.
 WITH SOMEWHAT THE SCORES $f(x, W) = W_x$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,

and using the shorthand for the
 scores vector:

$$s = f(x_i, W)$$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_j \geq s_{y_i} + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

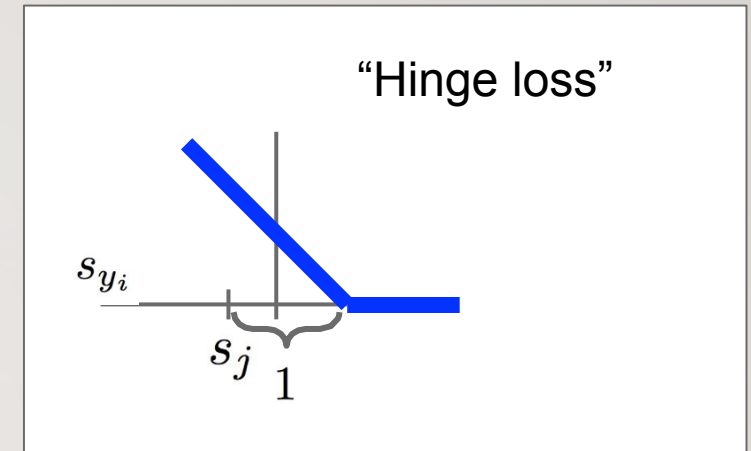
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

SUPPOSE: 3 TRAINING EXAMPLES, 3 CLASSES.
 WITH SOMEWHAT THE SCORES $f(x, W) = W_x$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

Multiclass SVM loss:



the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_j \geq s_{y_i} + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

SUPPOSE: 3 TRAINING EXAMPLES, 3 CLASSES.
 WITH SOME WEIGHTS THE SCORES $f(x, W) = W_x$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss:	2.9		

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,
 and using the shorthand for the
 scores vector:

$$s = f(x_i, W)$$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$\begin{aligned}
 &= \max(0, 5.1 - 3.2 + 1) \\
 &\quad + \max(0, -1.7 - 3.2 + 1) \\
 &= \max(0, 2.9) + \max(0, -3.9) \\
 &= 2.9 + 0 \\
 &= 2.9
 \end{aligned}$$

SUPPOSE: 3 TRAINING EXAMPLES, 3 CLASSES.
 WITH SOMEWHAT THE SCORES $f(x, W) = W_x$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss:		0	

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,
 and using the shorthand for the
 scores vector:

$$s = f(x_i, W)$$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$\begin{aligned}
 &= \max(0, 1.3 - 4.9 + 1) \\
 &\quad + \max(0, 2.0 - 4.9 + 1) \\
 &= \max(0, -2.6) + \max(0, -1.9) \\
 &= 0 + 0 \\
 &= 0
 \end{aligned}$$

SUPPOSE: 3 TRAINING EXAMPLES, 3 CLASSES.
 WITH SOME WEIGHTS THE SCORES $f(x, W) = W_x$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss:			12.9

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,
 and using the shorthand for the
 scores vector:

$$s = f(x_i, W)$$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$\begin{aligned}
 &= \max(0, 2.2 - (-3.1) + 1) \\
 &\quad + \max(0, 2.5 - (-3.1) + 1) \\
 &= \max(0, 6.3) + \max(0, 6.6) \\
 &= 6.3 + 6.6 \\
 &= 12.9
 \end{aligned}$$

SUPPOSE: 3 TRAINING EXAMPLES, 3 CLASSES.
 WITH SOMEWHAT THE SCORES $f(x, W) = W_x$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i)
 where x_i is the image and
 where y_i is the (integer) label,
 and using the shorthand for the
 scores vector:

$$s = f(x_i, W)$$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Loss over full dataset is average:

$$L = \frac{1}{N} \sum_{i=1}^N L_i$$

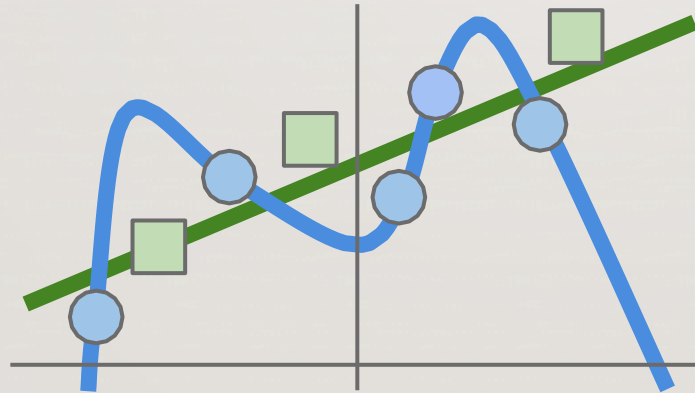
$$L = (2.9 + 0 + 12.9)/3 \\ = 5.27$$

OverFitting

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Model should be “simple”, so it works on test data

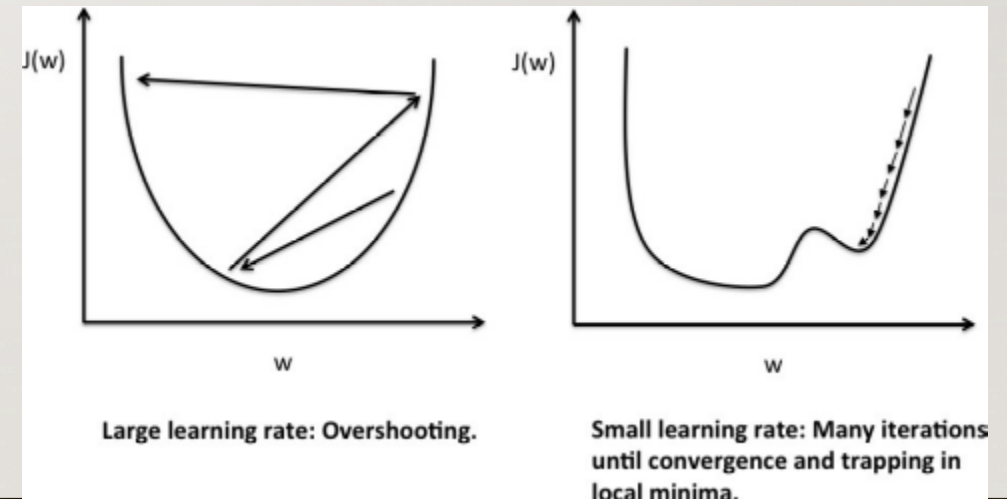
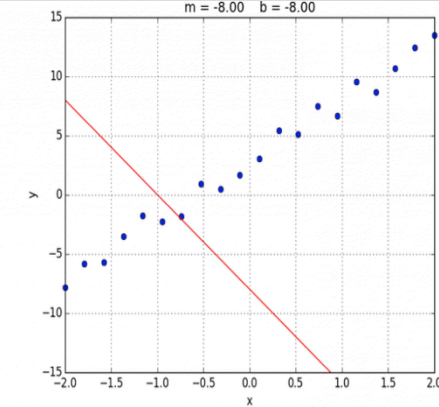
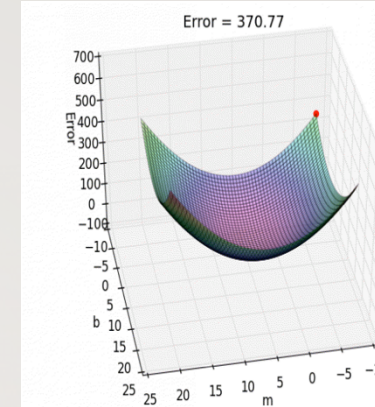


OPTIMIZATION

- Optimization Algorithms are used to update weights and biases i.e. the internal parameters of a model to reduce the error.

Gradient Descent

- Gradient descent is a way to minimize an objective function $J(w)$ parameterized by a model's parameters w
- It updates the parameters in the opposite direction of the gradient of the objective function w.r.t. to the parameters ($\nabla_w J(w)$)
- The learning rate η determines the size of the steps we take to reach a (local) minimum

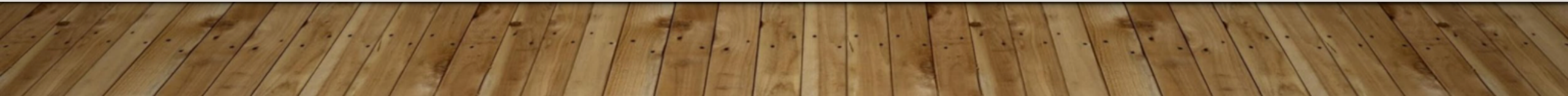


Gradient Descent

Vanilla Gradient Descent Algorithm:

- Start with an initial set of coefficients for the function
These could be 0.0 or a small random value.
 $\text{coefficient} = 0.0$
- Calculate the derivative of the cost. The derivative is a concept from calculus and refers to the slope of the function at a given point. We need to know the slope so that we know the direction(sign) to move the coefficient values in order to get a lower cost on the next iteration.
 $\text{delta} = \text{derivative}(\text{cost})$
- Now that we know from the derivative which direction is downhill, we can now update the coefficient values.
- A learning rate parameter (alpha) must be specified that controls how much the coefficients can change on each update.
 $\text{coefficient} = \text{coefficient} - (\text{alpha} * \text{delta})$
- This process is repeated until the cost of the coefficients (cost) is 0.0 or close enough to zero to be good enough.

NEURAL NETWORKS AND BACKPROPAGATION ALGORITHM

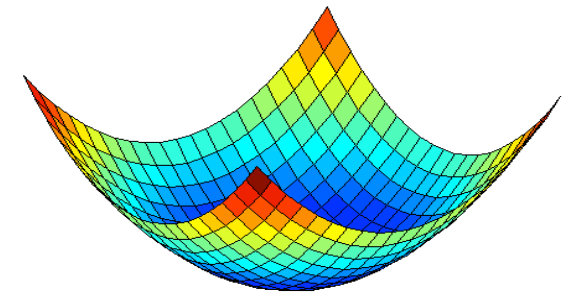
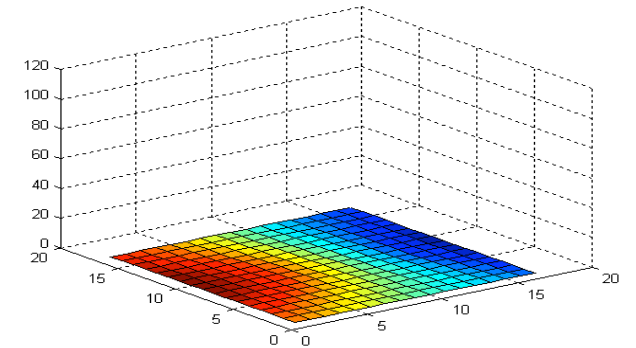


INTRODUCTION

- **Backpropagation** is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. It is commonly used to train deep neural network, a term referring to neural networks with more than one hidden layer.
- The term is an abbreviation for “backwards propagation of errors”.

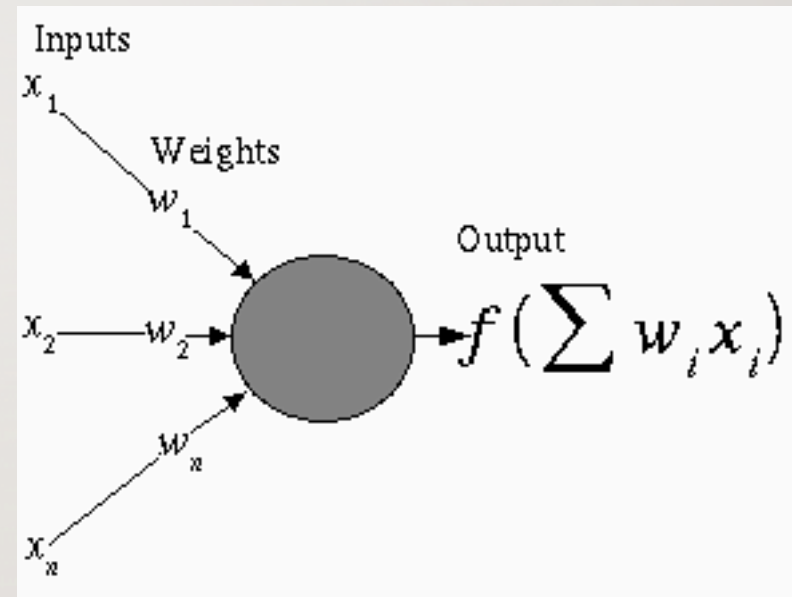
INTUITION

- As the algorithm's name implies, the errors (and therefore the learning) propagate backwards from the output nodes to the inner nodes.
- So technically speaking, backpropagation is used to calculate the gradient of the error of the network with respect to the network's modifiable weights.
- This gradient is almost always then used in a simple stochastic gradient descent algorithm to find weights that minimize the error. “



BASIC NEURON MODEL - FEEDFORWARD NETWORK

- **Inputs x_i** are fed through input connections
- Specific functions are modeled using real **weights w_i**
- The output of the neuron is a **nonlinear function f** of its weighted inputs



INPUTS TO NEURONS

- Arise from other neurons or from outside the network
- Nodes whose inputs arise outside the network are called *input nodes* and simply copy values
- An input may *excite* or *inhibit* the response of the neuron to which it is applied, depending upon the weight of the connection

Weights

- Normally, positive weights are considered as excitatory while negative weights are thought of as inhibitory
- **Learning** is the process of modifying the weights in order to produce a network that performs some function

Output

The response function is normally nonlinear

Samples include

- Sigmoid
$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$

- Piecewise linear

$$f(x) = \begin{cases} x, & \text{if } x \geq \theta \\ 0, & \text{if } x < \theta \end{cases}$$

BACKPROPAGATION PREPARATION

- **Training Set**
A collection of input-output patterns that are used to train the network
- **Testing Set**
A collection of input-output patterns that are used to assess network performance
- **Learning Rate- α**
A scalar parameter, analogous to step size in numerical integration, used to set the rate of adjustments

NETWORK ERROR

- Total-Sum-Squared-Error (TSSE)

$$TSSE = \frac{1}{2} \sum_{patterns} \sum_{outputs} (desired - actual)^2$$

- Root-Mean-Squared-Error (RMSE)

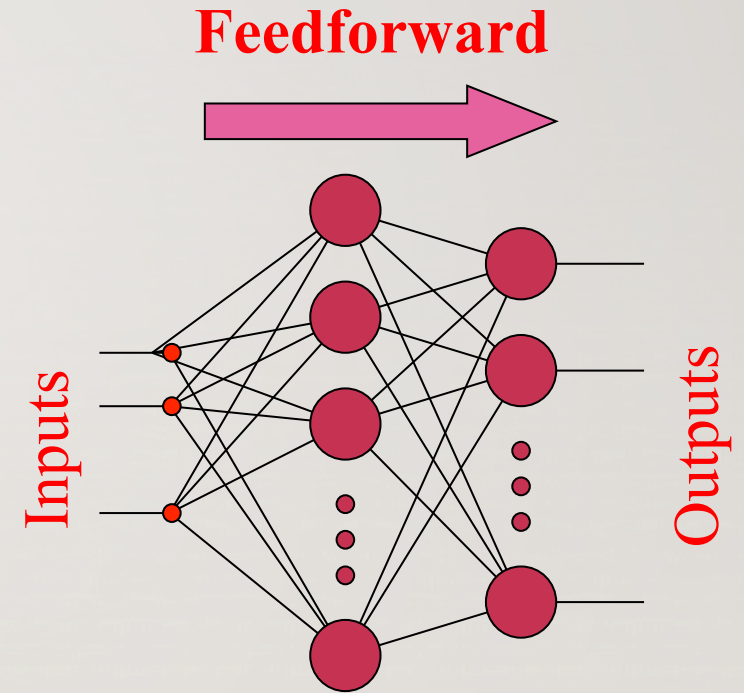
$$RMSE = \sqrt{\frac{2 * TSSE}{\# patterns * \# outputs}}$$

A PSEUDO-CODE ALGORITHM

- Randomly choose the initial weights
- While error is too large
 - For each training pattern (presented in random order)
 - Apply the inputs to the network
 - Calculate the output for every neuron from the input layer, through the hidden layer(s), to the output layer
 - Calculate the error at the outputs
 - Use the output error to compute error signals for pre-output layers
 - Use the error signals to compute weight adjustments
 - Apply the weight adjustments
 - Periodically evaluate the network performance

APPLY INPUTS FROM A PATTERN

- Apply the value of each input parameter to each input node
- Input nodes compute only the identity function



CALCULATE OUTPUTS FOR EACH NEURON BASED ON THE PATTERN

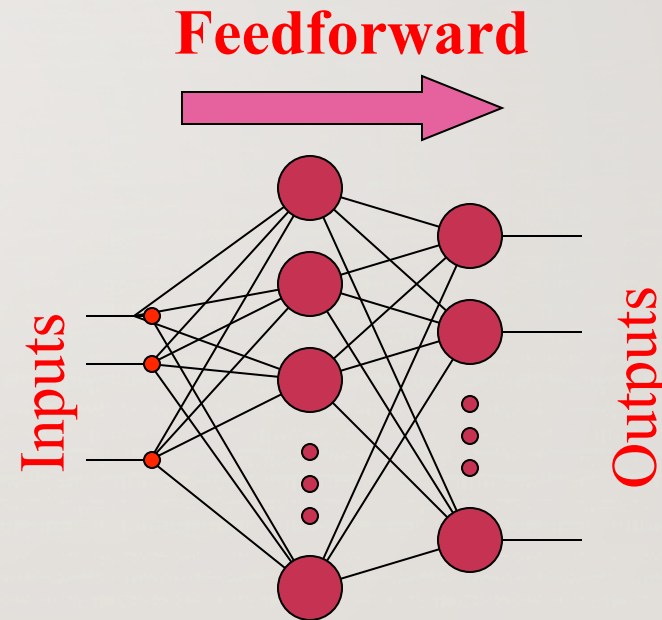
- The output from neuron j for pattern p is O_{pj} where

$$O_{pj}(net_j) = \frac{1}{1 + e^{-\lambda net_j}}$$

and

$$net_j = bias * W_{bias} + \sum_k O_{pk} W_{kj}$$

- k ranges over the input indices and W_{jk} is the weight on the connection from input k to neuron j



CALCULATE THE ERROR SIGNAL FOR EACH OUTPUT NEURON

- The **output neuron error signal** δ_{pj} is given by $\delta_{pj} = (T_{pj} - O_{pj}) O_{pj} (1 - O_{pj})$
- T_{pj} is the target value of output neuron j for pattern p
- O_{pj} is the actual output value of output neuron j for pattern p

Source:

<http://people.uncw.edu/tagliai/Courses/415/Lectures/An%20Introduction%20To%20The%20Backpropagation%20Algorithm.ppt>

<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

CALCULATE THE ERROR SIGNAL FOR EACH HIDDEN NEURON

- The **hidden neuron error signal** δ_{pj} is given by

$$\delta_{pj} = O_{pj} (1 - O_{pj}) \sum_k \delta_{pk} W_{kj}$$

where δ_{pk} is the error signal of a post-synaptic neuron k and W_{kj} is the weight of the connection from hidden neuron j to the post-synaptic neuron k

CALCULATE AND APPLY WEIGHT ADJUSTMENTS

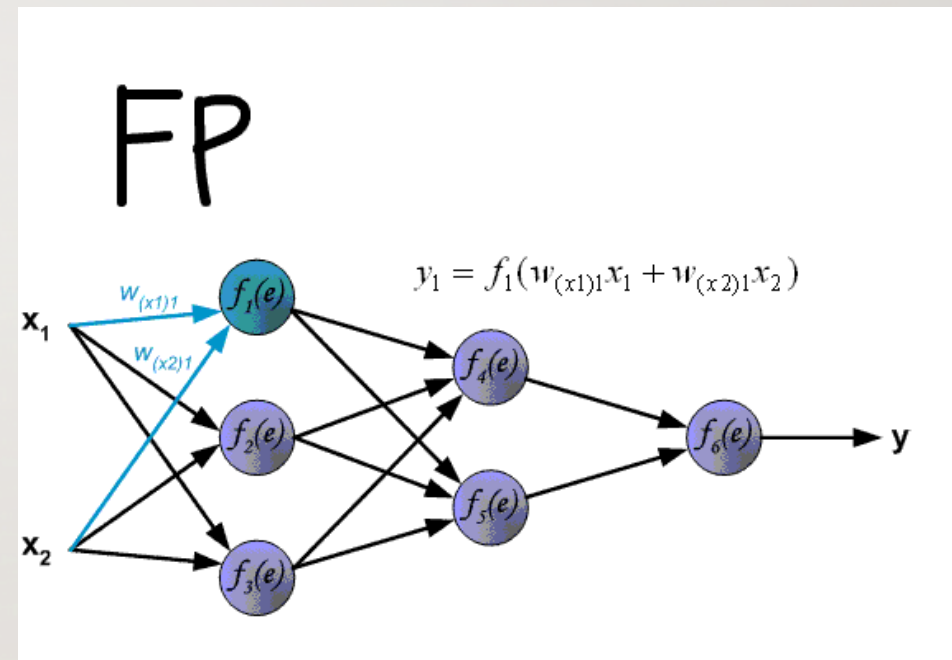
- Compute weight adjustments

ΔW_{ji} at time t by

$$\Delta W_{ji}(t) = \alpha \delta_{pj} O_{pi}$$

- Apply weight adjustments according to

$$W_{ji}(t+1) = W_{ji}(t) + \Delta W_{ji}(t)$$



MERITS AND DEMERITS OF BACKPROPAGATION

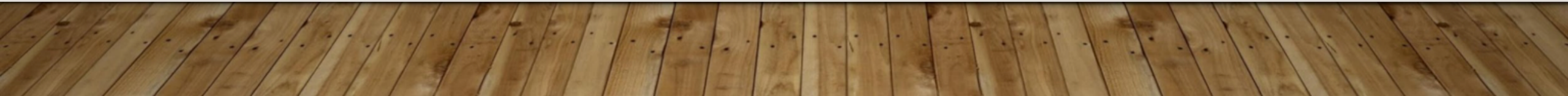
MERITS

- Relatively simple implementation.
- Mathematical Formula used in algorithm can be applied to any network. It does not require any special mention of the features of the function to be learnt.
- Batch update of weights exist, which provides a smoothing effect on the weight correction terms.

DEMERITS

- Slow and inefficient. Can get stuck in local minima resulting in sub-optimal solutions .
- A large amount of input/output data is available, but you're not sure how to relate it to the output.
- Outputs can be “fuzzy” or non-numeric.

CONVOLUTIONAL NEURAL NETWORK



WHY CNN?

- ConvNets are powerful due to their ability to extract the core features of an image and use these features to identify images that contain features like them.
- Even in a two layer CNN we can start to see the network paying a lot of attention to regions like the whiskers, nose, and eyes of the cat.
- These are the types of features that would allow the CNN to differentiate a cat from a bird for example.

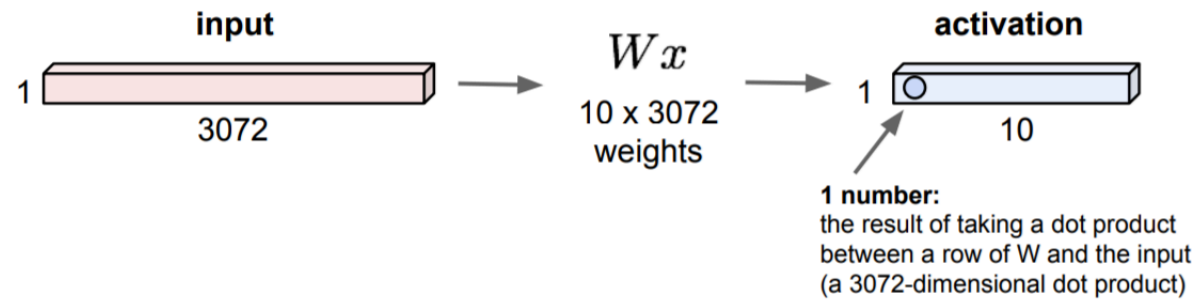
Source:

<https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59>

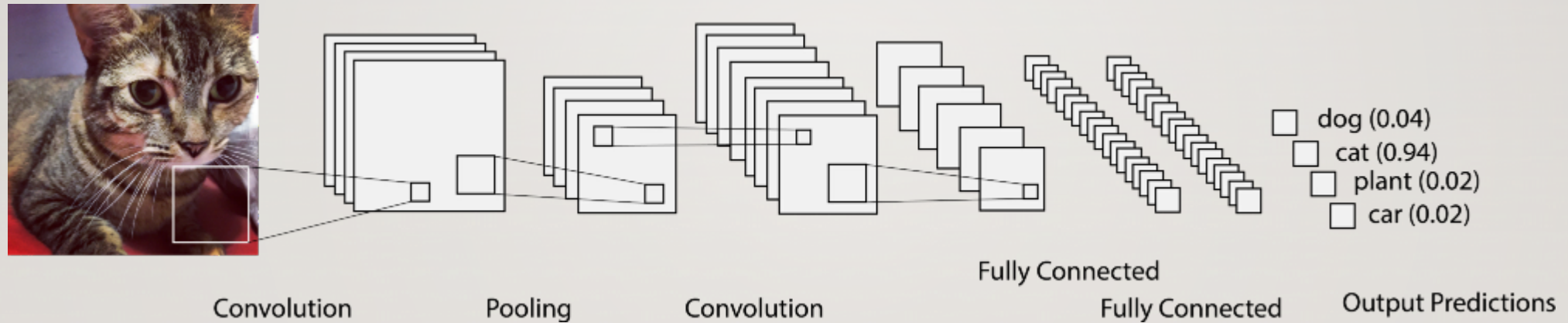
NEURAL NETWORK

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



CNN ARCHITECTURE



CNN LAYERS

- **Convolutional layers**
- Activation layers
- Pooling layers
- Fully Connected Layer

CONVOLUTIONAL LAYER

- The convolutional layer is the core building block of a CNN.
- The CONV layer's parameters consist of a set of learnable filters (Kernel).
- Conv layer maintains the structural aspect of the image
- As we move over an image we effectively check for patterns in that section of the image.
- When training an image, these filter weights change, and so when it is time to evaluate an image, these weights return high values if it thinks it is seeing a pattern it has seen before.
- The combinations of high weights from various filters let the network predict the

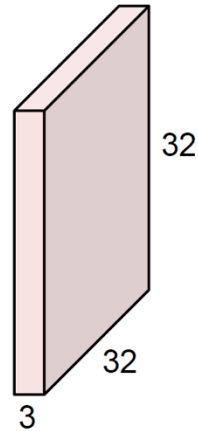
content of an image

Source: https://en.wikipedia.org/wiki/Convolutional_neural_network
<https://cs231n.github.io>
<https://medium.com/@Aj.Cheng/convolutional-neural-network-d9f69e473feb>

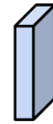
CONVOLUTED IMAGE

Convolution Layer

32x32x3 image



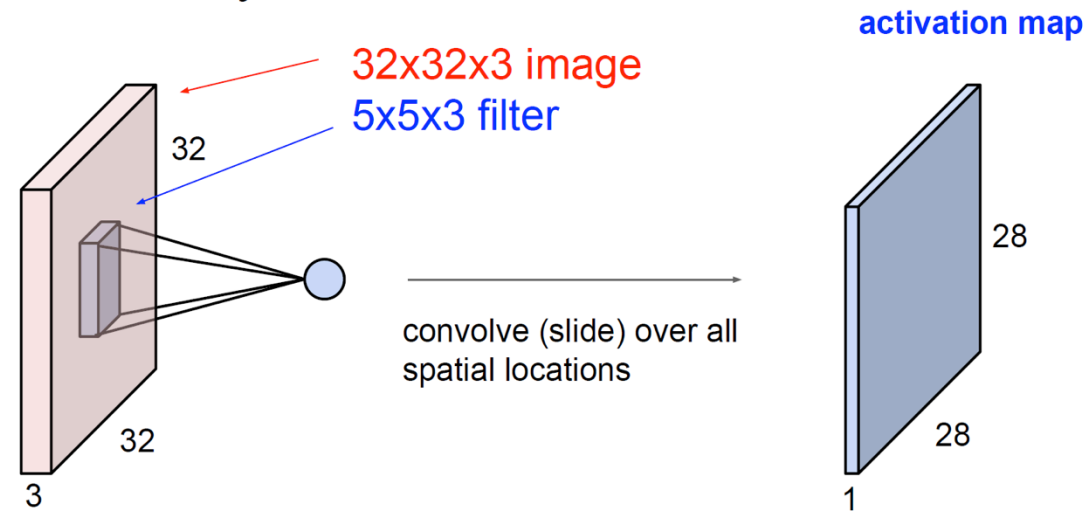
5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

CONVOLUTION

Convolution Layer



CONVOLUTION EXAMPLE

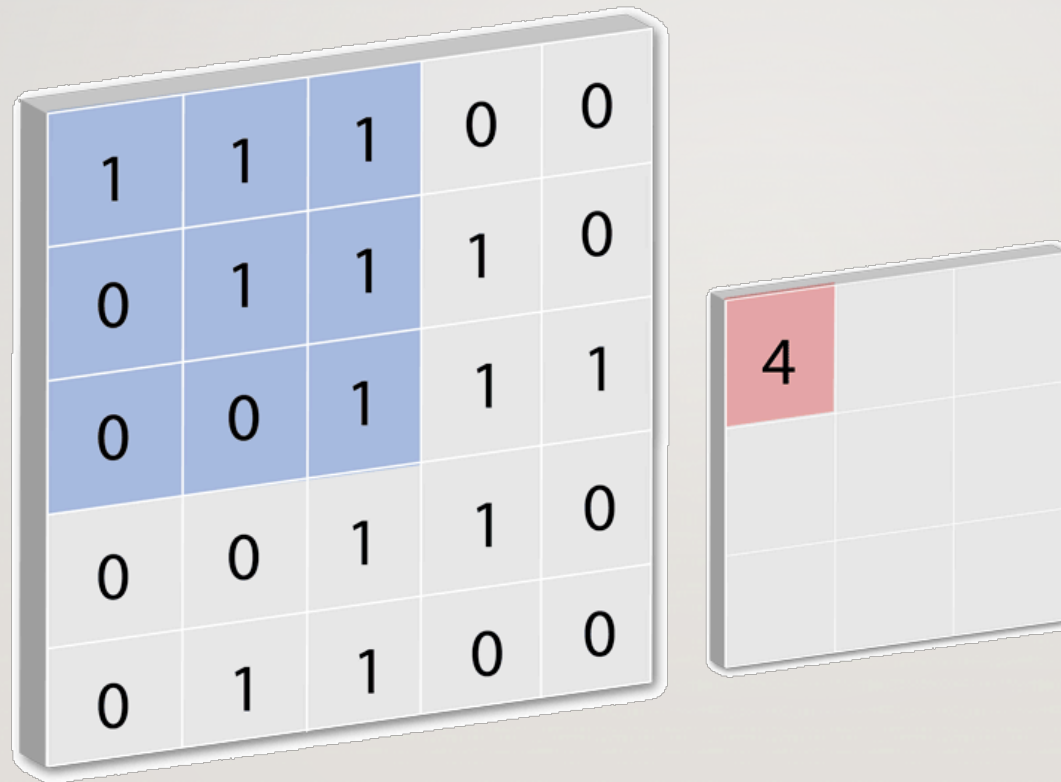
1	1	1	0	1
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

1	0	1
0	1	0
1	0	1

Convolved Feature

CONVOLUTION EXAMPLE

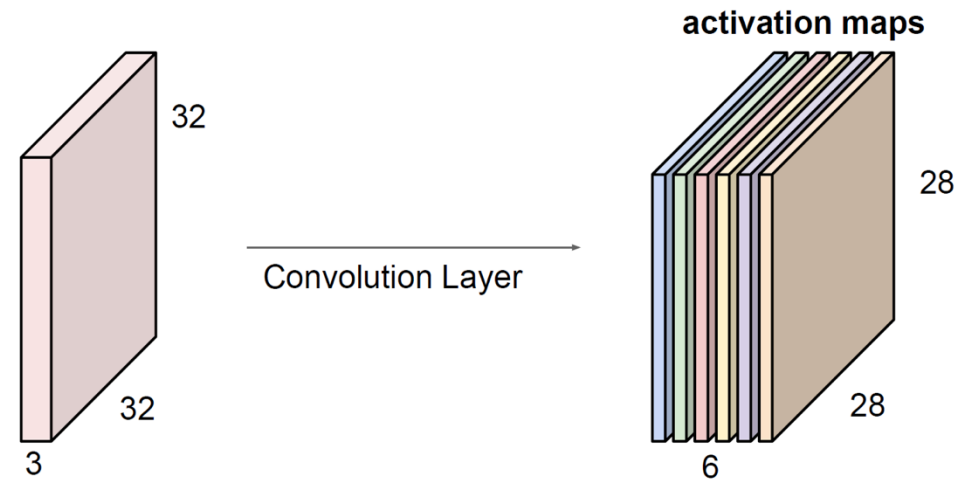


CONVOLUTION(IMPORTANT TERMINOLOGY)

- Stride: The distance the window moves each time.
- Kernel: The “window” that moves over the image.
- Depth: Depth of the output volume is a hyperparameter. It corresponds to the number of filters we would like to use, each learning to look for something different in the input.
- Zero-padding: Hyperparameter. We will use it to exactly preserve the spatial size of the input volume so the input and output width and height are the same

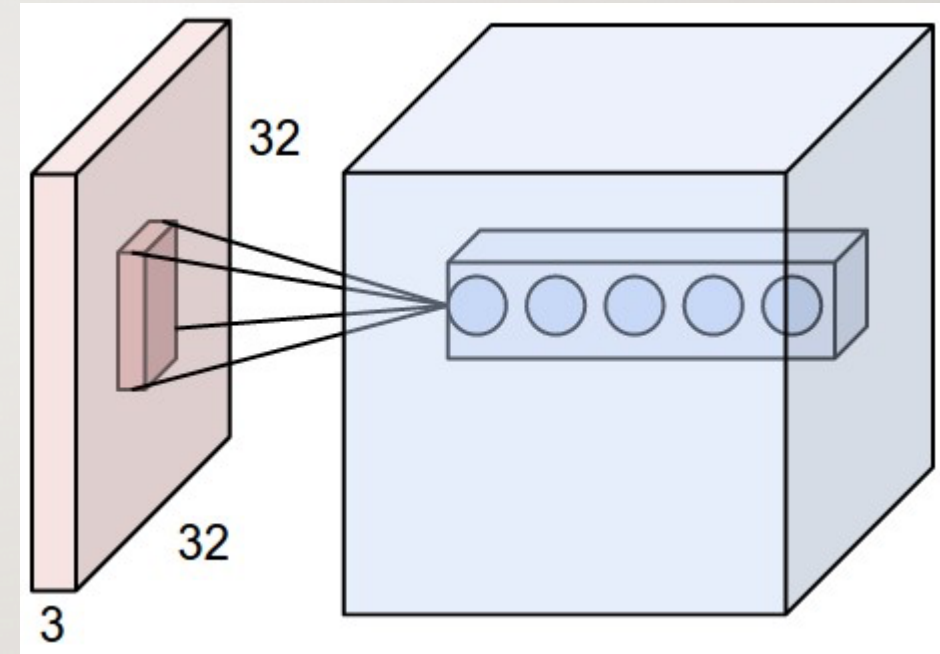
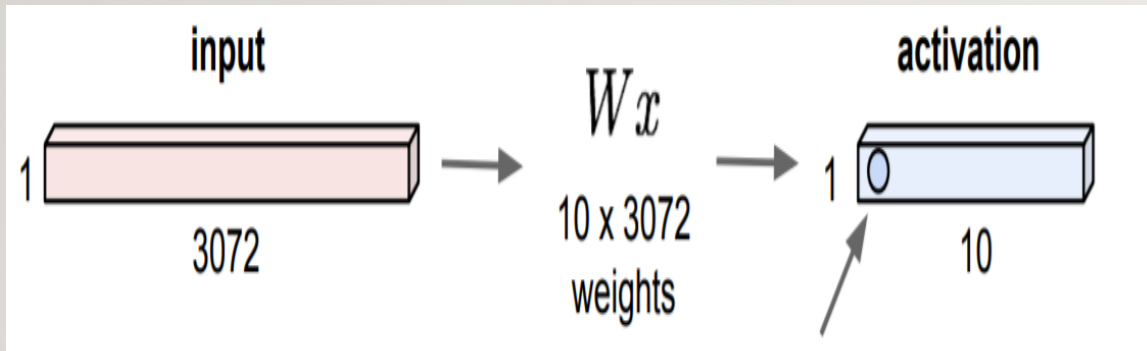
MULTIPLE FILTERS

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a “new image” of size 28x28x6!

SIMPLE FULLY CONNECTED NN VS CNN

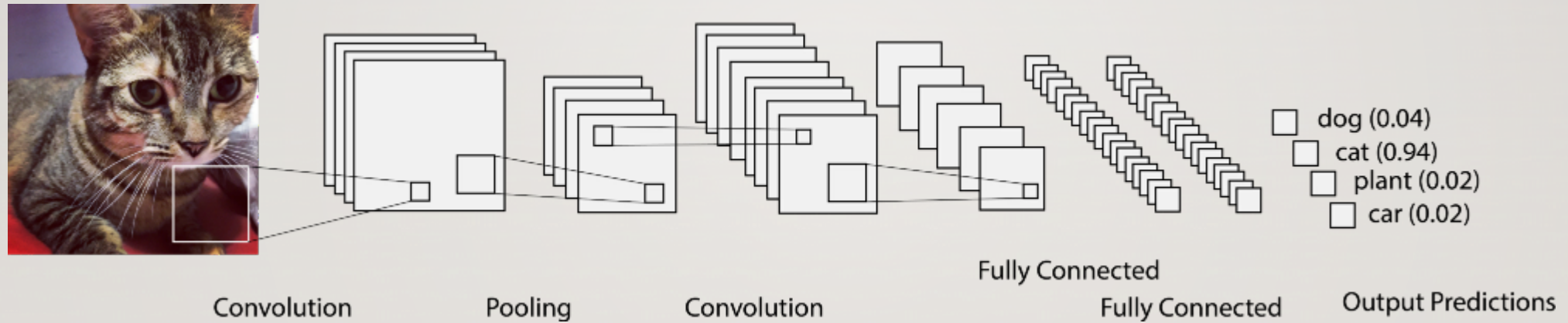


CNN retains the structure of the image

CNN LAYERS

- Convolutional layers
- **Activation layers**
- Pooling layers
- Fully Connected Layer




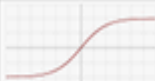


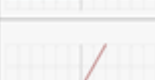
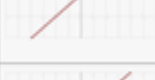

CNN ARCHITECTURE



ACTIVATION LAYER

- The purpose of the Activation Layer is to squash the value of the Convolution Layer into a range, usually $[0,1]$
- This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the convolution layer.
- Examples: tanh, sigmoid, ReLu

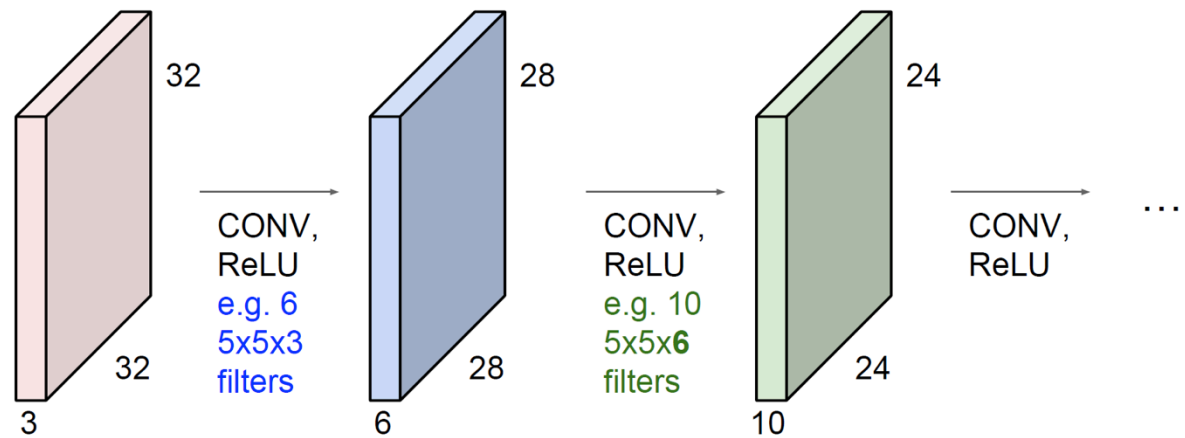
DIFFERENT ACTIVATION FUNCTIONS

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Source: <https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59>

MULTIPLE LAYERS OF CNN AND RELU

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Fei-Fei Li & Justin Johnson & Serena Yeung

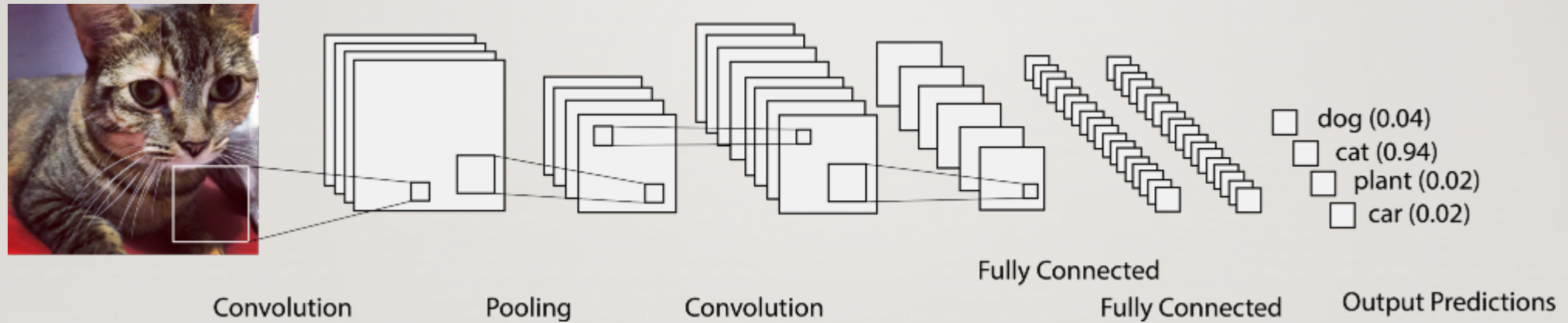
Lecture 5 - 36

April 18, 2017

CNN LAYERS

- Convolutional layers
- Activation layers
- **Pooling layers**
- Fully Connected Layer

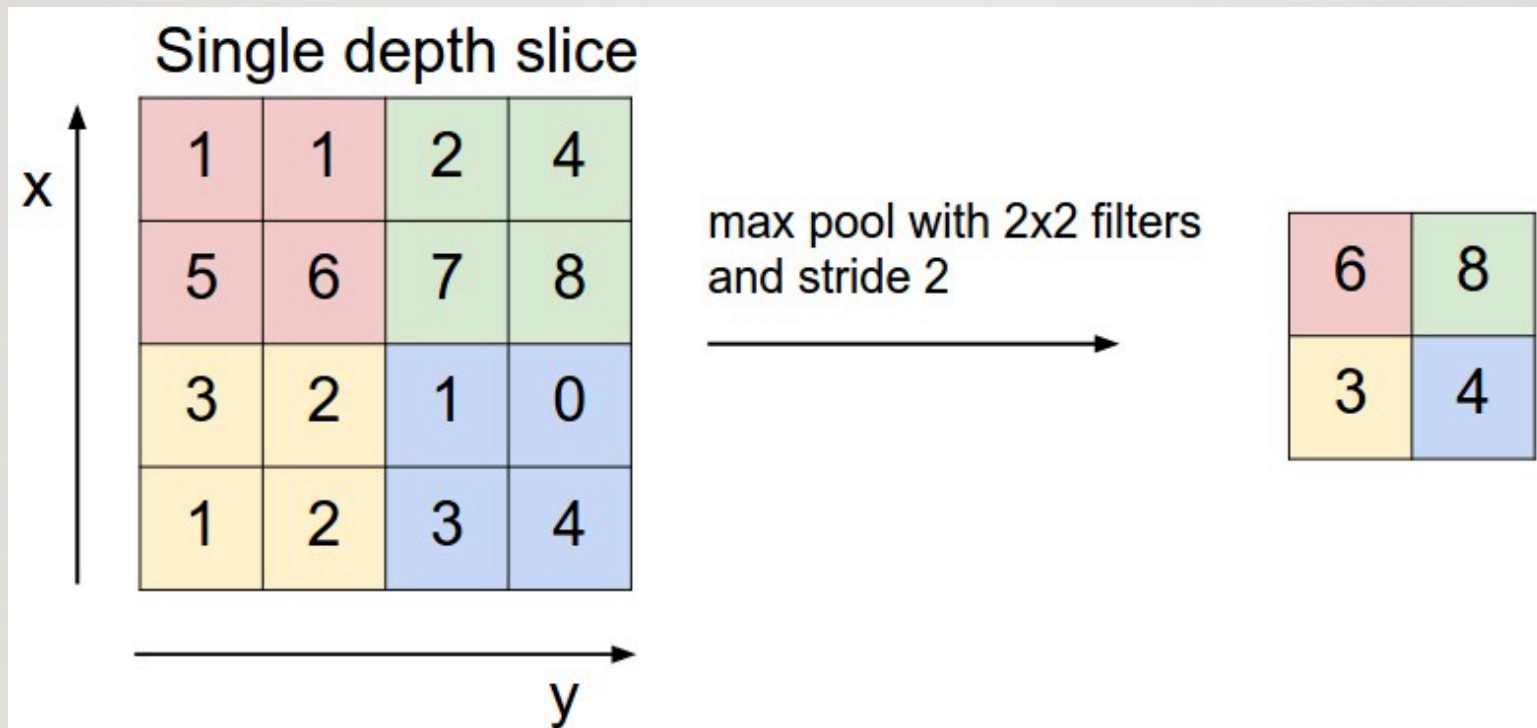
CNN ARCHITECTURE



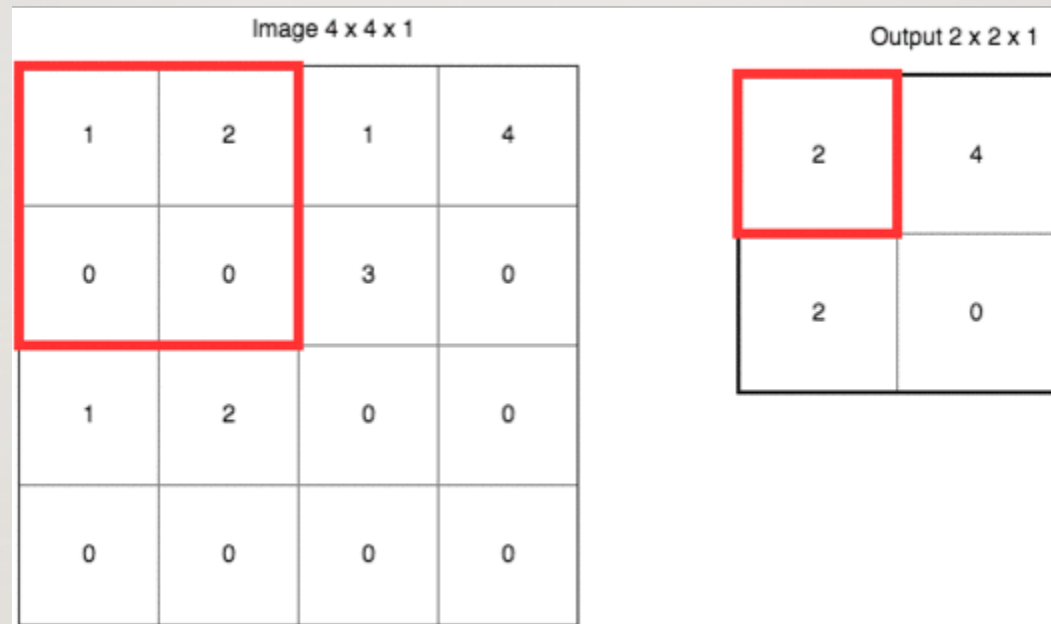
POOLING LAYER

- Pooling Layer's function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.
- **Max pooling** and **Average pooling** are the most common pooling functions. Max pooling takes the largest value from the window of the image currently covered by the kernel, while average pooling takes the average of all values in the window.

POOLING LAYER(MAX POOL)



POOLING LAYER(GRAPHICAL REPRESENTATION)



SUMMARY OF CNN LAYERS

- **Convolutional layers** multiply kernel value by the image window and optimize the kernel weights over time using gradient descent
- **Pooling layers** describe a window of an image using a single value which is the max or the average of that window(Max Pool vs Average Pool)
- **Activation layers** squash the values into a range, typically $[0,1]$ or $[-1,1]$.
- **Fully Connected Layer** Neurons have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

DEMO

- <https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

IMAGENET CLASSIFICATION WITH DEEP CONVOLUTIONAL NEURAL NETWORKS

By Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton

Journal: Advances in neural information processing systems (2012)

Outline

- Goal
- Dataset
- Architecture
- Overfitting
- Reducing Overfitting
- Results

ILSVRC: ImageNet Large Scale Visual Recognition Competition

- Annual competition of image classification at large scale
- 1.2M images in 1K categories
- Classification: make 5 guesses about the image label



Appenzeller

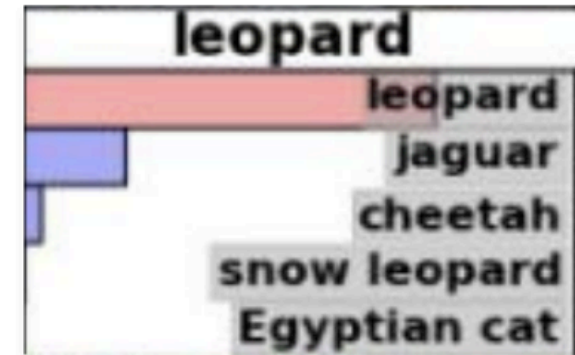


EntleBucher

Goal



Classification



DATASET

- The dataset used was a subset of ImageNet dataset with roughly 1000 images of each of the 1000 categories.
- In all, there were roughly,
 - 1.2 million Training images
 - 50,000 validation images
 - 150,000 test images

PREPROCESSING OF DATA

- The ImageNet consisted of variable-resolution images thus each image was downsampled to a fixed resolution of 256 x 256.
- Given a rectangular image, the image was rescaled such that the shorter side was of length 256, and then cropped out the central 256×256 patch from the resulting image.
- So the network was trained on (centered) raw RGB values of the pixels.

THE ARCHITECTURE

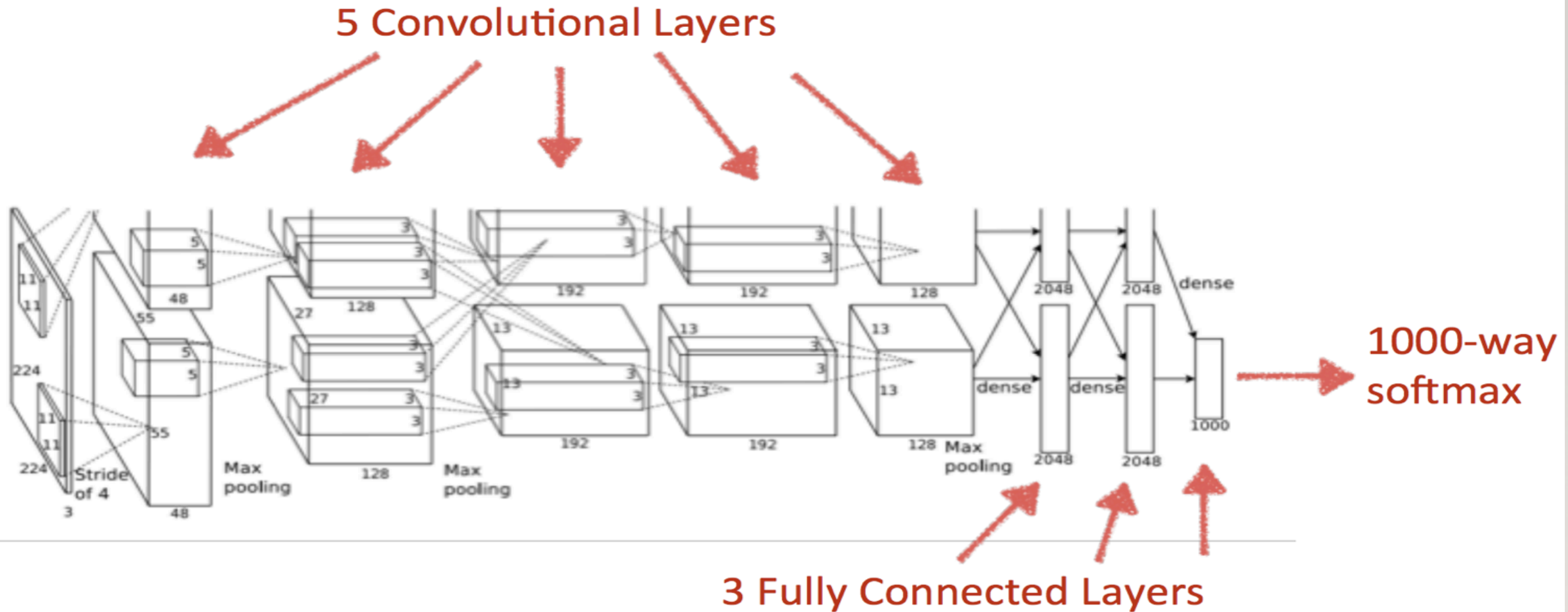
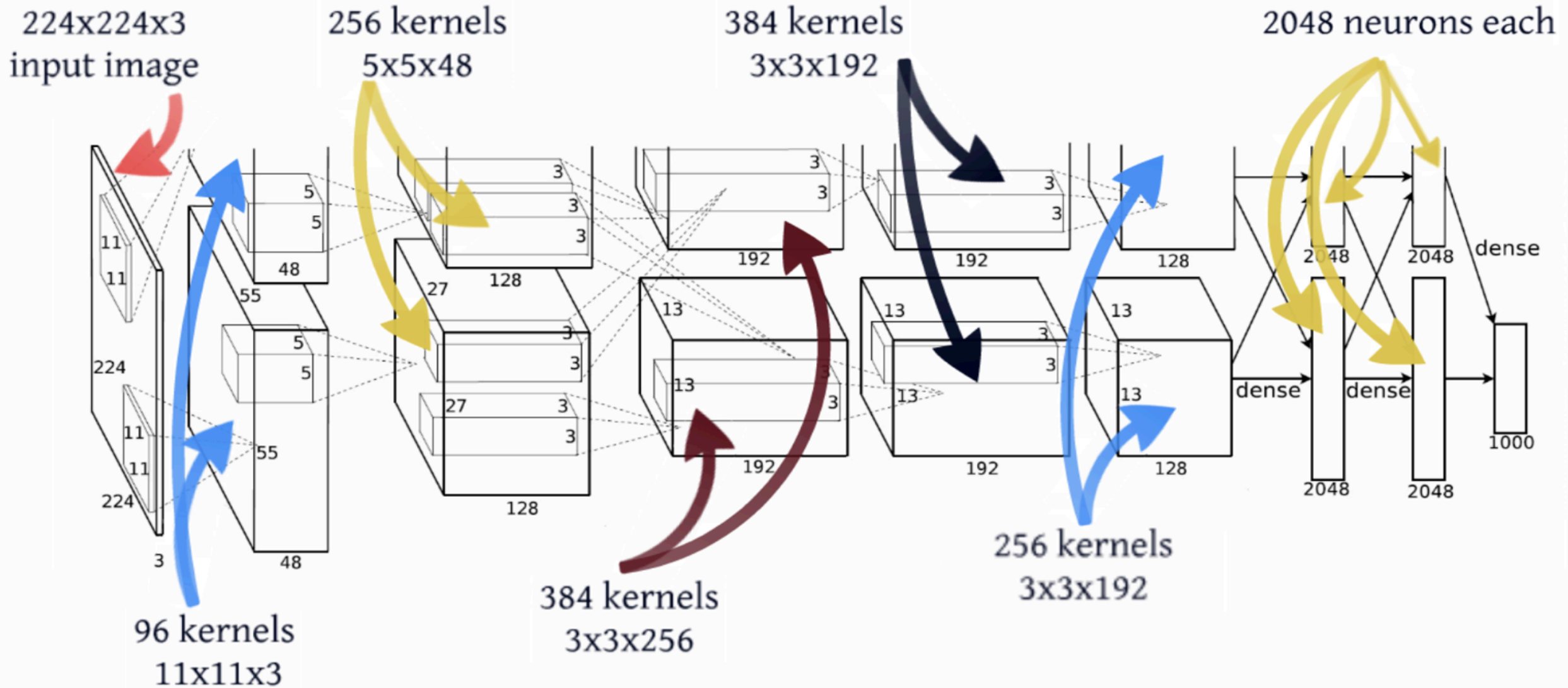


Image Source: Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

THE ARCHITECTURE

- The net contains eight layers with weights; the first five are convolutional and the remaining three are fullyconnected layers.
 - The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels.
- CONV1
 - MAX POOL1
 - NORM1
 - CONV2
 - MAX POOL2
 - NORM2
 - CONV3
 - CONV4
 - CONV5
 - Max POOL3
 - FC6
 - FC7
 - FC8

THE ARCHITECTURE

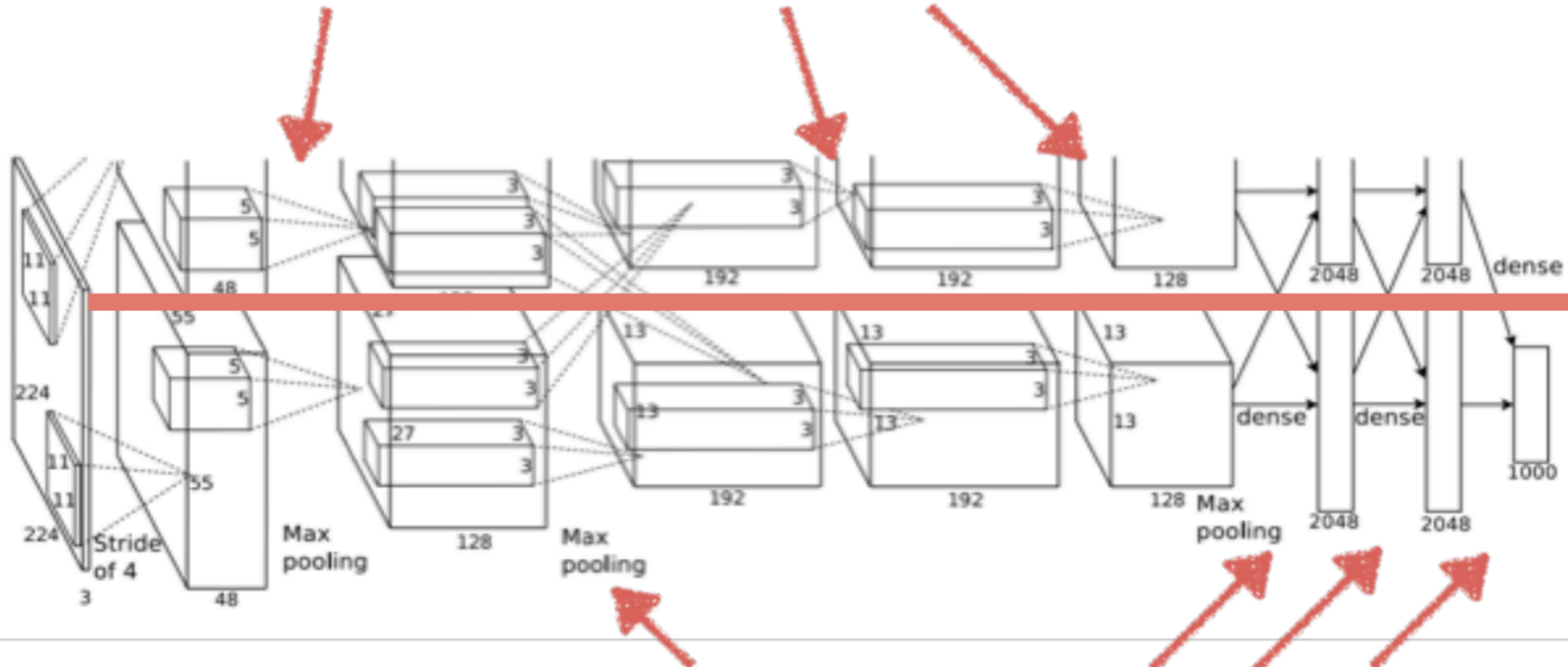


Source: Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

Training on Multiple GPUs

GPU #1

intra-GPU connections



GPU #2

inter-GPU connections

Overfitting

- 60 million parameters, 650,000 neurons
 - Overfits alot

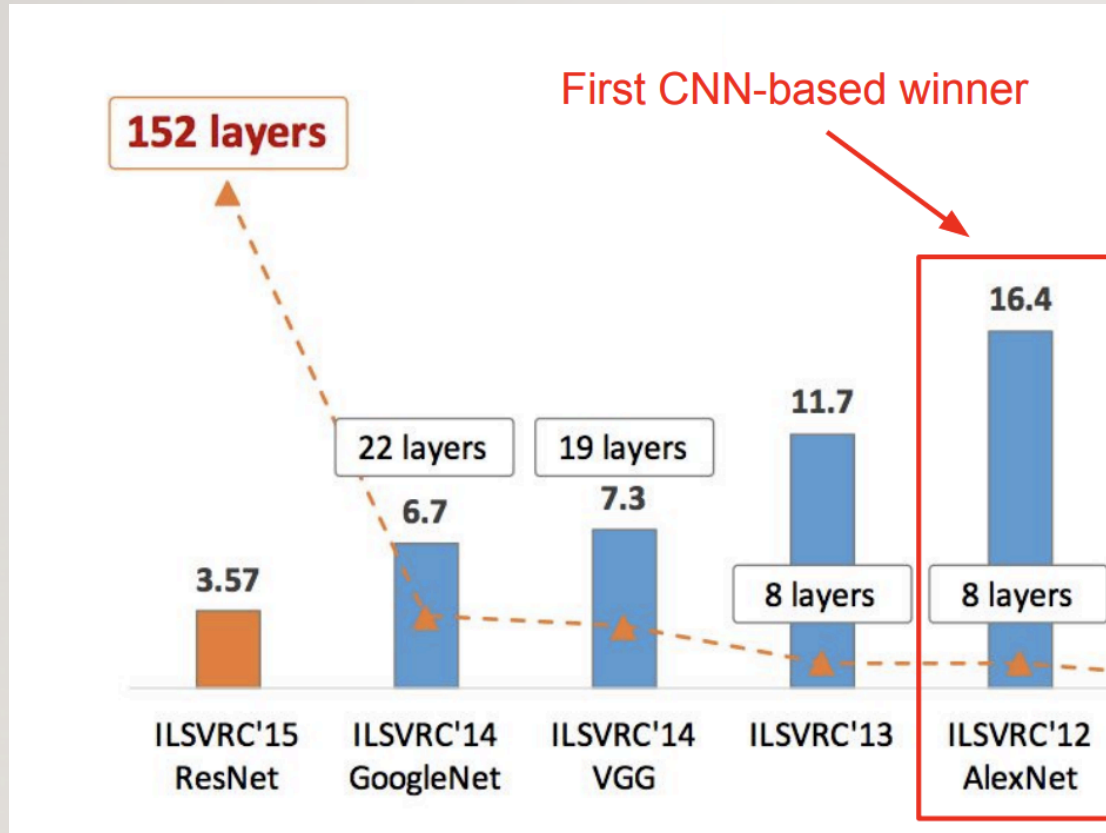
REDUCING OVERFITTING

- The focus of this paper was to reduce overfitting whilst outperforming state-of-the-art models.
- The two ways implemented to reduce overfitting were:
 - Data Augmentation
 - Dropout
- **Data Augmentation:** It is the process of artificially enlarging the dataset using label-preserving transformations. It was done in two ways:
 - Generated image translations and horizontal reflections. This is done by extracting random 224×224 patches (and their horizontal reflections) from the 256×256 images and training the network on these extracted patches. This increases the size of the training set by a factor of 2048.
 - Altered the intensities of the RGB channels in training images
- **Dropout:** It is a method of setting the output of each hidden neuron to zero with probability

Source: Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

overfitting.

RESULTS



- The network won the contest, achieving top-1 and top-5 test set error rates of 37.5% and 16.4%.